## REMARKS

The applicants have carefully considered the official action mailed on March 5, 2008, and the references applied therein. In the official action, claims 1-33 were rejected under 35 U.S.C. §102(b) as allegedly anticipated by Kim et al. (The Structure of a Compiler for Explicit *and* Implicit Parallelism).

By way of this response, claims 1, 12, 14, 19, 21, 23, 24, 29, 31, and 32 have been amended to include the subject matter from now canceled claims 11, 20, and 28, respectively. Accordingly, claims 1-10, 12-19, 21-27, and 29-33 remain pending in this application, of which claims 1, 19, 24, 31, and 33 are independent. Favorable reconsideration is respectfully requested in view of the foregoing amendments and the following remarks.

The applicants respectfully submit that independent claim 1 is allowable over the art of record. Independent claim 1 is directed to a method of compiling a program that, *inter alia*, determines a misspeculation cost value for at least one speculative parallel thread candidate, and selects a set of speculative parallel threads from a set of speculative parallel thread candidates based on a misspeculation cost value. Kim et al. do not describe or suggest determining a misspeculation cost value for at least one speculative parallel thread candidate, and selecting a set of speculative parallel threads from a set of speculative parallel thread candidates based on a misspeculation cost value, as recited in claim 1.

Instead, Kim et al. describe a compiler structure for implicit and explicit parallelism, where implicit parallelism refers to "extract(ing) parallelism speculatively from a sequential instruction stream" and explicit parallelism refers to "execut(ing) explicit parallel code sections as a multiprocessor." (See Abstract). However, the methods used by Kim et al. to identify explicit and implicit threads do not include determining a misspeculation cost value for at least one of a set of speculative parallel thread candidates, much less selecting a set of speculative parallel threads from a set of speculative parallel thread candidates based on a determined misspeculation cost value.

To the contrary, Kim et al. describe selecting each outermost parallel loop and determining whether to explicitly multithread the code or implicitly (speculatively) multithread the code based on estimated overheads at compile time. (See section 2.2, page 5, paragraph 1). Also, because Kim et al. only consider fully parallel loops for explicit threading (see Section 2.2, page 4, last paragraph), the explicit threading described by Kim et al. is not speculative and only the implicit loops are considered by Kim et al. to be speculative. This distinction between explicit and implicit (speculative) threading is also generally known in the art. The algorithm factors described by Kim et al. to select parallel loops for explicit threading include if a loop L is more than doubly nested; if the loop L encloses any serial loop; if the loop L contains any function calls; if the compiler detects any speculative storage overflow (i.e., an L1 cache conflict); or if the workload is greater than a threshold. (See Section 4, Algorithm 1).

First, if the workload for the loop L is greater than a threshold, or if the loop L is assumed to have a workload that is greater than the threshold due to more-than-double nesting, the loop L is executed as an explicit thread. (See page 10, 4th paragraph, *Loop Workload*). The workload is the projected benefit of executing a parallel thread and is not a misspeculation cost as recited in claim 1. Second, if the loop L encloses any inner, serial loop, the loop L is executed as an explicit thread. The system described by Kim et al. detects an inner, serial loop if there are any compiler-detected (i.e., automatically detected) cross-iteration dependencies. Determining the presence of inner-loop serial loops ensures a processor will not stall or hang while executing the program due to dependencies on variables that span multiple threads, and cannot be fairly construed as determining a misspeculation cost. Third, if the loop L contains any function calls, the loop L is executed as an explicit thread because implicit execution of the thread by a processor would incur stalls and hangs due to artificial dependencies of the speculative thread. Determining function calls also does not constitute determining a misspeculation cost. Finally, if a thread would incur speculative storage overflow (i.e., if the number of speculatively written values exceed the capacity of the speculative storage in the L1 cache) by executing an implicit thread, the loop L is executed explicitly. (See Section 4, page 10, paragraphs 2 and 3). Thus, none of the foregoing criteria described by Kim et al. could be interpreted to be a misspeculation cost as recited in claim 1.

Thus, because Kim et al. fail to describe or suggest determining a misspeculation cost value for at least one speculative parallel thread candidate and, therefore, fail to describe or suggest selecting a set of speculative parallel threads from a set of speculative parallel thread candidates based on a misspeculation cost value, Kim et al. necessarily fail to anticipate independent claim 1. The applicants respectfully request that the rejection of independent claim 1 be withdrawn for at least the foregoing reasons. Accordingly, the rejection of claims 2-10 and 12-18 dependent thereon must also be withdrawn for the foregoing reasons.

Independent claims 19, 24, and 31 are also allowable for reasons similar to those set forth above in connection with independent claim 1. In particular, each of claims 19, 24, and 31 is directed to an article of manufacture storing machine readable instructions, an apparatus, or a system that, *inter alia*, determines a misspeculation cost value for at least one speculative parallel thread candidate, and selects a set of speculative parallel threads from a set of speculative parallel thread candidates based on a misspeculation cost value. None of the cited references describes or suggests determining a cost value for at least one of a set of speculative parallel thread candidates, and selecting a set of speculative parallel threads from a set of speculative parallel thread candidates based on a cost value, as recited in claims 19, 24, and 31, and claims dependent therefrom. Accordingly, the applicants respectfully submit that claims 19, 24, and 31, and claims dependent thereon are believed to be in condition for allowance.

The applicants also submit that independent claim 33 is allowable over the art of record. Independent claim 33 recites, *inter alia*, determining a likelihood that a data dependency violation will occur, determining an amount of computation required to recover from the data dependency violation, and selecting at least one of a set of speculative parallel thread candidates based on a lowest likelihood of misspeculation. Kim et al. neither describe nor suggest determining a likelihood that a data dependency violation will occur, determining an amount of computation required to recover from the data dependency violation, and selecting at least one of a set of speculative parallel thread candidates based on a lowest likelihood of misspeculation, as recited in claim 33.

At best, the compiler described by Kim et al. is configured to operate based on assumptions that avoid data dependencies in speculative threads completely by executing parallel loops as explicit threads or attaching an "IF" statement with a condition that must be met to run the parallel loops as implicit threads. The examiner appears to contend that

Kim et al. describes determining a likelihood that the data dependency violation will occur, and determining an amount of computation required to recover from the data dependency violation. [*See page 5 of the official action*]. In particular, the examiner appears to contend that the first type of dependency described by Kim et al. constitutes determining a likelihood that a data dependency violation will occur, and determining an amount of computation required to recover from the data dependency violation. However, Kim et al. merely describe the register communication mechanism and synchronization used by previous compilers. In fact, Kim et al. do not provide any description or suggestion of determining an amount of computation required to recover from a data dependency, much less selecting at least one of the set of speculative parallel thread candidates based on a lowest likelihood of misspeculation, as recited in claim 33. While Kim et al. describes estimating overheads associated with explicit threading (See Kim et al., section 2.2, page 5, paragraph 1), Kim et al. fail to describe or suggest any determination of a computation required to recover from a data dependency, much less selecting at least one of the set of speculative parallel thread candidates based on a lowest likelihood of misspeculation.

Thus, for at least the foregoing reasons, the applicants respectfully submit that all pending claims are now in condition for allowance. If there are any remaining issues in this application, the applicants urge the examiner to contact the undersigned attorney at the number listed below.

The Commissioner is authorized to charge any deficiency in the enclosed check toward payment of any fee due for the filing of this paper to deposit account number 50-2455.

Respectfully submitted,

_/Mark G. Hanley/___________

Mark G. Hanley
Reg. No. 44,736
Attorney for Applicants
150 S. Wacker Drive, Suite 2100
Chicago, IL 60606
(312) 580-1020

**August 29, 2008**